**A05**

**Due Date:**

Thursday, June 1

**File to submit:**

TextItems.java

**Sample Run:**

[SampleOutput.html](http://cs.smu.ca/~myoung/csci2341/Assignments/A05/SampleOutput.html)

**Testing File:**

[TestMyTextItems.java](http://cs.smu.ca/~myoung/csci2341/Assignments/A05/TestMyTextItems.java)

**Data Files:**

[OneItem.txt](http://cs.smu.ca/~myoung/csci2341/Assignments/A05/OneItem.txt), [MiniItems.txt](http://cs.smu.ca/~myoung/csci2341/Assignments/A05/MiniItems.txt), [ManyItems.txt](http://cs.smu.ca/~myoung/csci2341/Assignments/A05/ManyItems.txt), [WithPauses.txt](http://cs.smu.ca/~myoung/csci2341/Assignments/A05/WithPauses.txt), [Truncated.txt](http://cs.smu.ca/~myoung/csci2341/Assignments/A05/Truncated.txt)

[SUBMIT](http://cs.smu.ca/~myoung/csci2341/protected/getSubmission.cgi?05)   /   [Check](http://cs.smu.ca/~myoung/csci2341/protected/viewasgn.cgi?A05)

**Creating the TextItems Class (File I/O, Exceptions, Lists)**

**Summary**

Implement the **TextItems** class. This class is designed to show text one page at a time. It consists of a constructor and a method to display an item. The text is drawn from a file when a TextItems object is constructed.

Use List objects to hold the information read from the file. Catch any exceptions that occur and respond with an appropriate message.

**Details**

The **TextItems** class encapsulates an object that holds and displays pre-formatted text. Each block of text has a name that the client uses to request the block, and it may include "pauses" where the computer will stop and wait for the user to press the enter key.

The text comes from a file, the name of which is given to the constructor. For example:

TextItems ti = new TextItems("MiniItems.txt");

The text in the file is in this format:

* The first line is the title of the text item.
* The following lines are the text of the item.
* After the last line of the item's text, there is a line of 80 hyphens, signaling the end of the item. (This must have *exactly* 80 hyphens; otherwise it's just another line in the file.)
* Items repeat in the same pattern until the end of the file.

For example, the file MiniItems.txt is as follows:

First Item

First line of first item

Second line of first item

--------------------------------------------------------------------------------

Second Item

First line of second item

Third line of second item

--------------------------------------------------------------------------------

It consists of two items, one named "First Item" and the other named "Second Item". The first item has two lines, while the second has three (the middle line being blank). When the client asks to display "First Item" (ti.displayItem("First Item");) the screen should show

First line of first item

Second line of first item

When the client asks for the "Second Item" the screen would show

First line of second item

Third line of second item

Some items will contain a line of 80 at signs (@). Those lines are not to be printed, but instead signify that the print-out should *pause* (with a brief message, such as "...press enter...") and wait for the user to press the enter key. (The line must have *exactly* 80 at signs or else it's just another line to be printed.) For full credit that's just what your code should do.

**Further Requirements**

* You must use **List** instance variables in your implementation. Use of other data structures/data types will be penalized.
* Use a private inner class to hold the data for a single text item.
* If the file requested by the client can't be found (or read from), your code should print an appropriate message on System.err and leave the object with no text items.
* If the file ends unexpectedly, your code should print an appropriate message on System.err and leave the object with no text items (even if some items had previously been successfully read).
* If the client should ask for a text item that doesn't exist, your code should print an appropriate error message on System.err.

**Hints**

* Remember that NetBeans looks for files starting in the project folder (not the src folder). Be sure to place your files there and *make sure they have the right names*(remember that Windows often lies to you about the names of files).
* **Work in small steps!** Recompile and run the test program after each step. Make sure it's working properly before you go on to the next step.
  1. Create stubs for the constructor and method.
  2. Add code to the constructor to open the file (or deal with failure).
  3. Add code to "scan" the file -- read in each item and print out its title without saving that information anywhere. Get it to scan one item then add the outer loop to read multiple items. (Make sure that it prints out all the titles from each file.)
  4. Add code to fill in the text items' lines.
  5. Add code to deal with truncated data files (it's another catch block for the try-catch block you should already have).
  6. Add code to print out an item -- not worrying about the pauses.
  7. Add code to pause at the appropriate times.
  8. Do a thorough testing, including making sure that BOTH "fails" at the end do what they're supposed to.
* **Checking for item end/pause** You can create variables to hold Strings with 80 hyphens and exclamation marks, and compare the Strings you read to them, but there is an easier way. It's possible to ask a String if it matches a *pattern*, and one pattern you can specify is a number of repetitions of a single character. For example,

line.matches("%{20}")

return true if (and only if) line is a String of exactly 20 % signs. Similarly,

str.matches("E{50}K!")

return true if (and only if) str is a String of exactly 50 Es followed by a K and an exclamation point. From these I trust you can figure out how to ask if a String is exactly 80 hyphens/at signs long.

[SUBMIT](http://cs.smu.ca/~myoung/csci2341/protected/getSubmission.cgi?05)   /   [Check](http://cs.smu.ca/~myoung/csci2341/protected/viewasgn.cgi?A05)